16302010002 李云帆

hw3

我在本次作业中，用`hadoop`实现了一下对`sample.txt`的一些文本处理，用的是java语言。

任务1

统计其中各类文件的数量（按文件名后缀区分类型）

项目路径：**filecount_test_2019_3_28/**

我写的程序`wordcount.java`中，基本实现思路和上次作业的`wordcount`一样，只是这次的数据中不做所有词的词频统计，有比较多的冗余信息，我的方法是只挑出文件后缀名进行统计。（

```
//map方法的重写，将数据拆分成<word,one>的形式，将数据以<Text,IntWritable>的形式传送到reduce
public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
    StringTokenizer it = new StringTokenizer(value.toString(), ".\n\r");
    while (it.hasMoreTokens()) {
        String nextToken = it.nextToken();
        //由于文件中后缀名长度都不超过5，因此在这里做个判断，只选后缀名做count
        if (nextToken.length() > 5) {
            continue;
        }
        word.set(nextToken);
        context.write(word, one);
    }
}
```

任务2

按文件的字节数大小降序排序输出文件名

项目路径：**fileNameSort/**

我写的`fileOP.java`中，由于`hadoop`默认是对`key`做升序排序输出，而我们的要求是倒序，所以先写了一个

```
private static class DecreasingComparator extends LongWritable.Comparator
//继承LongWritable.Comparator，对key进行倒序
```

然后，在处理字符串输入的过程中由于是中文会产生乱码，而hadoop又不支持UTF-8，所以用GBK解析：

```
String line = new String(value.getBytes(), 0, value.getLength(), "GBK");
```

用了比较麻烦的方式对字符串做处理，浪费了很多时间，深感数据格式 **干净** 的重要性

这次的`mapper`和`reducer`都需要不一样的设计，

```
//继承Mapper，对数据进行拆分，其中<Object,Text>为数据的输入类型，<LongWritable, Text>为数据的输出类型
public static class fileNameMapper extends Mapper<Object, Text, LongWritable, Text>
```

还有一个对排序任务比较重要的：

```
//由于要按照文件大小降序，所以将文件大小作为key（hadoop里是按照key排序的）
context.write(new LongWritable(fileSize), fileType);
```

而reducer就比较简单暴力了，因为没什么需要reduce的

```
//继承Reducer，通过shuffle阶段获得Map处理后的<fileSize,fileType>的值，对数据直接以<fileSize,fileType>的形式输出
//其中<LongWritable, Text>为输入的格式,<LongWritable, Text>为输出的格式
public static class IntSortReducer extends Reducer<LongWritable, Text, LongWritable, Text>
```

reduce()里也是

```
context.write(key, value);
```

这里倒不是因为排序需要，只是图个方便啦，怎么输入就怎么输出

最后在

```
    public static void sortFileBySize(Path input, Path output, Configuration conf)
            throws IOException, InterruptedException, Exception {
//        Path tempDir = new Path("wordcount-temp"); //定义一个临时目录

        //建立job任务
        Job job = Job.getInstance(conf, "file sort by size");
        //配置job中的各个类
```

```
        job.setJarByClass(fileOP.class);
        job.setMapperClass(fileNameMapper.class);
        //combine方法是在reduce之前对map处理结果的一个局部汇总，一般有几个map就会有几个combine

        job.setCombinerClass(IntSortReducer.class);
        job.setReducerClass(IntSortReducer.class);
        //设置输入输出格式
        job.setOutputKeyClass(LongWritable.class);
        job.setOutputValueClass(Text.class);
        //用DecreasingComparator实现倒序
        job.setSortComparatorClass(DecreasingComparator.class);
        //设置输入输出路径
        FileInputFormat.addInputPath(job, input);
        FileOutputFormat.setOutputPath(job, output);
        //由任务是否完成而选择是否正常退出程序
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
```

里面需要设置一些参数来限定一些路径或者数据结构或者如何（倒序）输出

这个里面还有一些`java`的知识，比如`java`文件编译后变成`class`文件，`class`文件又可以打包成`jar`

在终端(terminal)中，比如：

```
javac fileOP.java;
jar cvf fileOP.jar *.class;
hadoop fs -rm -r countfile/output;
hadoop jar fileOP.jar fileOP /user/root/sample.txt /user/root/countfile/output
```

然后这次的作业就完成啦～

附

要求的输出文件为`filecount_test_2019_3_28/sampleoutput-filecount2/part-r-00000` & `fileNameSort//sampleoutput-filecount/part-r-00000`